

UNLIMITED

BR114203 (2)

Report No. 90001



Report No. 90001

ROYAL SIGNALS AND RADAR ESTABLISHMENT,
MALVERN

DTIC FILE COPY

AD-A225 639

THE TERRY-WISEMAN SECURITY
POLICY MODEL AND EXAMPLES
OF ITS USE

Author: C L Harrold

DTIC
ELECTE
AUG 22 1963
S B D
Co

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

March 1990

UNLIMITED

90 08 20 185

0074114

CONDITIONS OF RELEASE

BR 114203

.....

DRIC U

COPYRIGHT (c)
1988
CONTROLLER
HMSO LONDON

.....

DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Report 90001

Title: The Terry-Wiseman Security Policy Model and Examples of Its Use
Author: C L Harrold
Date: March 1990

Abstract

This paper presents a model of security for computer systems where it is essential that the confidentiality of the information is maintained. The model is introduced using a simple analogy, it is then defined formally and its use illustrated by examples. The framework for using the model to achieve high assurance implementations of high functionality systems is also discussed.

Contents

1. Introduction
2. An Analogy to Introduce the Model
3. The Formal Model and its Interpretation
 - 3.1. Informal Overview
 - 3.2. Set Definitions
 - 3.3. The State
 - 3.4. Supporting Definitions
 - 3.5. State Transitions
 - 3.6. The Security Axioms
4. Examples of the Use of the Model
 - 4.1. A User Interface: Windows
 - 4.2. Contents of Objects
 - 4.3. Regradable Documents
 - 4.4. Journals and Audit
 - 4.5. Drafting Documents
 - 4.6. A Multi-level Directory
 - 4.7. Access Control Lists
5. The Terry-Wiseman Approach to High Assurance Implementations
6. Evolution of the Model
7. Summary and References

Annex: An Overview of the Z Notation

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	Special
A-1	

1. Introduction

In order to have even a reasonable level of assurance that a system intended to be secure, is indeed so, there needs to be an "unambiguous, complete and consistent" statement of what the term 'security' means in that particular context. In other words, there must be a statement of security, which is in some sense valid, against which to assess the system produced.

Determining the validity of any statement of security requires that all concerned, for example the customers, developers and evaluators, interpret it in the same way and agree that the requirements have been met. To this end, a model of security must be neither so abstract nor so mechanistic that it restricts users requirements. The Terry-Wiseman Security Model and modelling approach strikes a balance between these two extremes. Although Terry-Wiseman has been developed primarily for SMITE [Harold89,Bottomley88,Wiseman86], we believe that the approach is more generally applicable.

The main problem area that Terry-Wiseman addresses is that of maintaining the confidentiality of information, in other words ensuring that unauthorised people cannot discover classified information. The model is powerful enough to model all aspects of a computerised system including human interaction. This is because, in addition to providing the simple 'no downward flows' property, as provided by the Bell and LaPadula [Bell74] and non-interference [Goguen82] models, the Terry-Wiseman Model describes the controls on creation and deletion and also on alterations to the controls themselves. That is, the model uses separation of duty [Linden76,Clark&Wilson87] to provide integrity on the controls used to enforce confidentiality.

The basic Bell and LaPadula model is more mechanistic than Terry-Wiseman as it is concerned with the ability to access rather than the actual accesses to objects. Non-interference, on the other hand, is more abstract as it is in terms of the use of interfaces to the system. However, it is not the place of this paper to give detailed comparisons, which can be found in [Terry&Wiseman89]. In addition, [Terry90] gives a non-interference proof of the no flows down property of the Terry-Wiseman Model using m-Eves [Craig87].

This paper introduces the Terry-Wiseman security model in section 2 using a simple analogy. The formal model is presented, in section 3, using a mathematical notation and is accompanied by its interpretation. Section 4 shows how the model can be used to describe a wide variety of security features. Section 5 provides an overview of the role of the model in the development of high assurance, high functionality systems. Section 6 discusses the differences between the model in this paper and the original formalisation of the Terry-Wiseman Model [Terry&Wiseman89].

2. An Analogy to Introduce the Model

This section provides an overview of the Terry-Wiseman Model, describing the area the model is addressing, the concepts and assumptions of the model and the level of abstraction, using a simple analogy. In this analogy information is hidden in *black boxes*. These boxes have a slot through which further information may be added and a drawer from which information may be retrieved. There are also buttons, levers, etc, with which to ask for information, add information, and so on. The boxes are black so that nothing whatsoever about their contents is visible from the outside.

In addition to the various levers and buttons on the outside of the box there are a number of *labels*. One of these says what the classification of the information inside the box is. The classification effectively defines who can receive information from the box plus who may add information to it. The use of the other labels will be identified as the analogy progresses.

These classified black boxes form the main mechanism for protecting the confidentiality of information. Obviously the introduction of any protection mechanism introduces signalling channels through the controls of the mechanism, because controls may always be probed and 'something' about them discovered. Consequently, the system ensures that the existence of the black boxes and the text written on labels convey no classified information.

The *system owners* are the organisation to whom the classified information ultimately belongs, but they delegate to other *people* to carry out the actual work. However, in order to limit the potential damage they could cause, the system owners will assign each person a *clearance* and a specific *role* (or roles). A person's clearance limits what they can see and their role what they can change.

There are three assumptions made about the people who use the system. The first is that the system owners gave them appropriate clearances. Secondly, it is assumed that people do not drive signalling channels, even if they have been subverted. We believe that this is true, as people have much better and faster ways of stealing the information, such as remembering it, copying it, and so on. Lastly, although some people might collude to defraud the systems owners, these possible collusions will have been anticipated by the systems owners in the way they have assigned the roles. What has to be assumed is that those people who it was decided would not collude, do not in fact do so. In other words, the systems owners are assumed to be competent and to have correctly identified the threats against their system.

However, in a computerised system it is not people who directly manipulate information but software that they run. Thus in the analogy the people who use the system ask *robots* to perform tasks for them. Some of these robots are expensive models and can be trusted to faithfully do what they were told, and not to do anything else. Other, cheaper, robots may not be so trusted. These may forget what they were told to do, do something different instead of, or in addition to, what they were told, get inquisitive about the contents of the black boxes and even actively attempt to communicate classified information to the unauthorised.

The robots have memory and are effectively mobile black boxes. They will have a classification label at, or below, the clearance of the person they are working for. Also, they will be labelled with the identity and role(s) of the person. Although some of the robots are honest, ie trusted to faithfully do exactly what they are told, this is not sufficient as the robots are able to give other robots tasks, or there may be groups of robots cooperating to carry out a task. Security critical operations require there to be honest robots whose orders can be traced back, through other honest robots, to a person. Such robots have a label indicating this trust.

A black box will only supply its contents to a robot whose classification label dominates the classification label of the box. Therefore, these classification labels are responsible for upholding confidentiality, and consequently their integrity is important. Integrity in Terry-Wiseman terms simply means that all the black boxes and robots must have a classification label in order that the confidentiality control may be applied, and that the classifications written on the labels must be appropriate. The appropriateness of the initial label given to a black box is the responsibility of its creator, but there is a requirement for some form of control on the alteration of labels.

It is generally too restrictive to simply prohibit all alterations of the labels, as this seriously restricts the possible functionality. It is not sufficient for the system to insist that labels may only be altered by robots faithfully carrying out the bidding of a person. This is because, although people can be trusted not to use devious and tedious signalling channels to steal information, they could easily be tempted to simply modify the labels so that they can see the information in the box, or they could simply make a mistake.

Therefore, the control placed on the alteration of labels is that a sufficient number of people (ie robots faithfully acting on their behalf) must agree that the change is necessary and appropriate, and further, that these people are not colluding to defraud the system owners. Thus, separation of duty is enforced on changing the controls. One of the labels of a black box will say which particular roles have to agree to any changes, and, as this label also requires integrity, it is governed by the same controls on its alteration.

Two signalling channels exist through which a rogue robot may attempt to leak information. The first is through the existence of black boxes. This potential signalling channel is stopped by simply making sure that the creators of black boxes are trusted not to signal information through their existence. In other words boxes may only be created by an honest robot carrying out a person's wishes, as described above.

The second potential signalling channel is through the labels placed on the outside of the black box. This signalling channel is stopped by only allowing the robots to create boxes, or modify the labels of existing boxes, if they are trusted not to explicitly write or encode classified information into the label. As above, these robots must be faithfully carrying out a person's wishes. In addition to being trusted not to pass information through the labels, the creator of a black box must be trusted to label new boxes appropriately.

The people tell the robots which black boxes to manipulate by specifying their *address*. Each black box therefore has a label on which is written its unique address, and the robots may observe this to check that they have found the right box. Note that as far as confidentiality is concerned it does not matter which boxes the robots manipulate so long as they are suitably cleared. Of all the aspects of security, it is only the separation of duty control that is concerned with which particular boxes are involved.

As confidentiality does not depend on the addressing mechanism, it would at first appear that addresses for objects should be part of the contents of the black boxes. However, this would make it impossible for the robots to check that they were standing in front of the correct box without effectively opening the box and comparing addresses, which they might not be cleared to do.

Thus, confidentiality is maintained by hiding information in classified boxes. Controls are put in place to ensure that boxes are suitably labelled at all times, ie the controls have integrity. Although the model is able to notice any changes to the protected information and can ensure that confidentiality was upheld, the integrity of such changes is an application specific issue, and can be safely ignored at this, higher, level of abstraction.

Thus, in summary, the information inside a black box is modelling the classified information in the memory of a computer. The files, objects or register locations in which the information may be stored are modelled by the black boxes. The robots model the computer software which manipulates this information. Some software is guaranteed to be functionally correct and free from additional Trojan Horse functionality, and can be modelled by the faithful, honest robots, who do not exploit signalling channels and whose actions can be traced back to the wishes of an authorised human user of the system.

3. The Formal Model and Its Interpretation

Following a brief informal overview, this section formally presents the model using the mathematical specification language Z [Spivey88]. An overview of the notation used in this paper is given in the annex. Z is based on set theory and uses boxes, called schemas, to structure the specification. In addition, a major part of any Z specification is explanatory English, and therefore this section need not to be omitted by those simply unfamiliar with the notation.

3.1 Informal Overview

The Terry-Wiseman Security Model considers that the state of any computer system can be captured by the relationships between sets of *entities* and *attributes*. Entities represent the containers of information, and are alterable objects in that their contents may change. Attributes are immutable objects and represent the information itself. Relating this to the analogy of the preceeding section, the robots are active entities, the black boxes passive entities and the information written on labels and stored inside boxes are attributes.

Confidentiality is concerned with controlling the flows of information. In the model information is said to flow between entities whenever a state transition occurs, that is whenever the relationships between entities and attributes are altered in some way. Furthermore, state transitions are modelled as having been initiated by a set of entities, called the *requestors*, rather than a single entity, in order to capture the notion of separation of duty. In the black box analogy, information may be said to flow whenever a robot puts information into a box, takes information out, alters any of the labels or when boxes are created or destroyed.

The objects of interest in any system may be described in terms of entities and attributes. Some of the attributes are security relevant, for example the classification written on a label, and relations and functions are defined on the state which explicitly identify these interesting attributes of an entity. The security requirements can then be captured by modelling secure transitions, that is defining the ways in which entities may gain or lose attributes and the conditions under which they may be created or destroyed.

3.2 Set Definitions

First the sets of all possible entities and attributes are introduced.

$[E, A]$

Next, a subset of all possible attributes is identified as representing classifications. It is also necessary to bring in the notions of the dominance relationship, \geq , between classifications, and the least upper bound, *LUB*, and greatest lower bound, *GLB*, operators on sets of classifications. As these concepts are well understood, the actual definitions have been omitted. Thus, the classification of an entity will be represented by an attribute drawn from the set called *CLASS*. Note that in some cases this may be interpreted as a clearance.

$CLASS : \mathbb{P} A$

$_ \geq _ : CLASS \leftrightarrow CLASS$

$GLB, LUB : \mathbb{P} CLASS \rightarrow CLASS$

A further subset of attributes are identified as representing the various types of *TRUST* that can be given to active entities. The three particular types of *TRUST* attribute, outlined in section 2, are identified.

The *dont signal* attribute will be given to those entities which are assumed not to exploit signalling channels, or otherwise to write classified information into the controls. The basis of this assumption may be that a human is in control, or that some other controls are being deployed to close the channel.

The *faithful* attribute will be given to entities which always do the bidding of a human user, in other words the human user is accountable for the actions of these entities, whether directly

through a trusted path, or indirectly because the software will always make the same decision/actions as the human. Faithful entities must be both functionally correct and free from Trojan Horses, ie proven to meet their specification.

Finally, the *creator* attribute will be given to those entities which are trusted to correctly set up the security controls on any new entities. The basis of this trust is application specific.

TRUST : $\mathbb{P}A$

dont_signal, *faithful*, *creator* : *TRUST*

The set of all possible roles and the unique identifiers of the human users of the system are identified as further subsets of all possible attributes. No specific roles are identified here as these are application specific. However, particular examples could be security officer, librarian, bank manager, counter clerk or system owner.

ROLE : $\mathbb{P}A$

ID : $\mathbb{P}A$

A further subset of attributes is identified to represent all possible conflicts, ie the n man rules, for the separation of duty control outlined in the analogy. A function is defined, *conflict_roles*, which gives the numbers of each type of role required. For example, the conflict attribute of a document could require one security officer and one ordinary user to agree to any alteration in its controls. All CONFLICT attributes uniquely have such numbers and roles associated with them.

CONFLICT : $\mathbb{P}A$

$\text{conflict_roles} : \text{CONFLICT} \rightarrow (\text{ROLE} \rightarrow \mathbb{N}_1)$
$\{ \} \notin \text{rng conflict_roles}$

Finally the set of attributes which represent references to entities, ie the means of addressing entities, is identified.

REF : $\mathbb{P}A$

3.3 The State

The state of the machine is given by the following schema, which structures the state into a number of named functions and relations between entities and attributes.

<p><i>STATE</i></p> <p><i>Class</i> : $E \rightarrow \text{CLASS}$</p> <p><i>Trust</i> : $E \leftrightarrow \text{TRUST}$</p> <p><i>Role</i> : $E \leftrightarrow \text{ROLE}$</p> <p><i>Id</i> : $E \rightarrow \text{ID}$</p> <p><i>Conflict</i> : $E \leftrightarrow \text{CONFLICT}$</p> <p><i>Ref</i> : $E \rightarrow \text{REF}$</p> <p><i>Other</i> : $E \leftrightarrow A$</p>
--

Class is the partial function which supplies the classification attribute of an entity. It is a function (many-to-one) as entities may only have a single classification and partial as nothing is specified about those entities which do not exist in the current state, ie those yet to be created or which have been destroyed.

Trust is the relation which supplies the various types of trust invested in an active entity. It is a

relation (many-to-many) as entities may have more than one trust attribute, and several entities may be similarly trusted.

Role is the relation which supplies the particular role (or roles) that an active entity plays in the system.

Id is the function which supplies the unique identity of the human user that an active entity is representing. It is a function as entities may represent at most one human, and partial as not all entities are active and so that entities may be both created and destroyed.

Conflict is the relation which identifies the various numbers and types of roles that must cooperate in any alteration to the security controls of an entity. It is a relation so that more than one conflicting set may be specified. Note that a single role is permissible even though it does not provide for any conflict of interest. This allows for an application to have very privileged and trusted entities, for example the system owners themselves. Also, if the requirement is that the controls of an entity are unalterable this should not be specified as no roles but as the set of all possible roles. This is because an empty set could allow unfaithful entities acting on their own to alter the controls.

There is one conflict set which covers modifications to all the security controls. However, there is no reason why a slightly different model could not be specified, with different conflicts for different controls, if this was the functionality required.

Ref supplies the means to address an entity, ie its reference. This is a function as each entity has a single reference, and partial to allow entities to be created and destroyed. The function is injective (one-to-one) as references may only be associated with a single entity, thus making references unique.

Other encompasses all the functionality aspects of the system, and is simply a relation between entities and attributes. Further, it is the attributes in the range of this relation which are considered to be protected by the controls. Therefore, this model is of all possible applications where confidentiality is the prime concern.

Note that the control type attributes may be found in the functionality attributes, for example, entities may contain references to other entities. Further, the various sets of attributes are not necessarily disjoint, and therefore a particular attribute may actually represent different things depending on context. For example, should integers be used to represent both classifications and months of the year, then the number 1 may mean Top Secret when it is representing the classification of an entity or January when part of the functionality.

3.4 Supporting Definitions

In defining the security policy model a number of functions are needed to characterise transitions in terms of the differences between states.

The following schema defines the symmetric set difference operator, \uparrow . This supplies all the differences between two sets. A second operator, \downarrow , is defined. This turns a relation into a function giving a set.

$[X, Y]$
$(\uparrow) : (\mathbb{P} X \times \mathbb{P} X) \rightarrow \mathbb{P} X$ $\downarrow : (X \leftrightarrow Y) \rightarrow (X \rightarrow \mathbb{P} Y)$
$\forall x, y : \mathbb{P} X \bullet x \uparrow y = (x \cup y) \setminus (x \cap y)$ $\forall r : X \leftrightarrow Y; x : \text{dom } r \bullet \text{dom } (\downarrow r) = \text{dom } r$ $\downarrow r(x) = r[\{x\}]$

The following schema defines an operator, *flatten*, which returns all the entity-attribute

relationships that comprise a state. All the entities that exist in a state may then be discovered, using the operator *entities*, which applies the domain operation to a flattened state.

$\text{flatten} : \text{STATE} \rightarrow (E \leftrightarrow A)$ $\text{entities} : \text{STATE} \rightarrow \mathbb{P} E$
$\forall s : \text{STATE} \bullet \text{flatten } s = s.\text{Class} \cup s.\text{Trust} \cup s.\text{Role} \cup s.\text{Id} \cup s.\text{Conflict} \cup s.\text{Ref} \cup s.\text{Other}$ $\text{entities } s = \text{dom}(\text{flatten } s)$

3.5 State Transitions

State transitions are considered to be requested by a set of entities, called the *requestors*. This is a set, rather than a single entity, in order to capture the notion of separation of duty for integrity. The second set of entities identified in the request are those entities which were *observed* during the transition. An entity is observed if its contents (ie the attributes defined by the *Other* relation) had any influence over the outcome of the state transition. It is very important that in an implementation entities not identified as observed have no influence over the outcome of a state transition. This observed set is identified in the transition request because although it is possible to examine the state to identify the receivers of information flow, ie those entities which had attributes changed, gained or lost, it is impossible to identify the source of the flow in the same way.

R $\text{requestors, observed} : \mathbb{P} E$
--

Note that the requestors of a state transition need not be observed. For example a command line interpreter decides what action to take on the basis of human input rather than its state. Also, the requestors need not be modified by the transition. Each requestor may say yes or no to the change as appropriate but will not necessarily remember whether all concerned agreed and the transition took place or not. For example, security officers may not remember whether they authorised a particular downgrade, although they may be able to look up the fact in a journal at a later date.

Valid state transitions are given by the following schema, which identifies the request and the before and after states. There are two constraints put on a valid state transition. The first is that transitions only occur if they are requested by entities that exist in the current state, and secondly, that the observed entities also exist.

TRANSITION $r? : R$ $s, s' : \text{STATE}$
$\{\} \subseteq r?.\text{requestors} \subseteq \text{entities } s$ $r?.\text{observed} \subseteq \text{entities } s$

3.6 The Security Axioms

This section formally defines security axioms to provide the confidentiality and integrity controls that were outlined in the analogy of section 2.

Confidentiality is divided into two separate concerns. The most obvious aspect of confidentiality is that information is not moved from highly classified to lowly classified containers. Relating this to the world of people looking at documents, for example, the windows of users' displays, which will be labelled with their clearance, cannot gain attributes from a

document classified higher than that clearance. Similarly in the analogy, a robot cannot discover the contents of a black box which has a label higher than its own.

The following schema, *no_flows_down*, identifies the set of entities whose view of the state was in some way modified. This view excludes the control attributes, as other axioms ensure that the controls on an entity are not classified. It also excludes entities which were destroyed, but does include newly created entities. These *modified* entities are therefore the receivers of the information flow. The source of the flow is simply those entities whose controls were observed.

Thus the axiom simply states that after the transition the lowest classification of the modified entities must dominate the highest classification of observed information before the transition. Thus information cannot flow down into either existing or newly created entities. Note that information may flow unconstrained between entities without classifications, and hence it is necessary to ensure that all entities have a classification, see the *Correctness* aspect of integrity below.

<i>no_flows_down</i>
TRANSITION
$GLB\ s'.Class \llbracket modified \rrbracket \geq LUB\ s.Class \llbracket r?.observed \rrbracket$
where
$modified == dom(s.Other \uparrow s'.Other) \cap entities\ s'$

The second aspect of confidentiality is that signalling channels are not exploited by untrusted software, formally expressed by the *no_signalling* axiom below. The signalling channels identified as *changed_controls* capture the channels through changing any of the security controls on existing entities and also through the creation and deletion of entities. The non-exploitation is expressed by insisting that should any of these aspects of the state be altered by a transition, then all the requestors must possess the *dont_signal* trust attribute.

<i>no_signalling</i>
TRANSITION
$changed_controls \neq \{ \} \Rightarrow (\forall r : r?.requestors \bullet dont_signal \in s.Trust \llbracket \{r\} \rrbracket)$
where
$changed_controls ==$ $\begin{aligned} & dom(s.Class \uparrow s'.Class) \\ & \cup dom(s.Trust \uparrow s'.Trust) \\ & \cup dom(s.Role \uparrow s'.Role) \\ & \cup dom(s.Id \uparrow s'.Id) \\ & \cup dom(s.Conflict \uparrow s'.Conflict) \\ & \cup dom(s.Ref \uparrow s'.Ref) \\ & \cup (entities\ s \uparrow entities\ s') \end{aligned}$

Thus, the confidentiality of information can be expressed as the conjunction of the above two aspects.

Confidentiality $\triangleq no_flows_down \wedge no_signalling$

In this model of security the prime concern is the confidentiality of information. Consequently the integrity of the controls that enforce confidentiality is also of concern, while as stressed earlier, the integrity of the protected information is an application specific issue. Integrity is divided into two aspects. Firstly, entities must have the necessary control attributes in order that the confidentiality controls may be applied, and secondly these controls must be in some way appropriate to the information they protect.

The *Correctness* schema simply insists that in order for any transition to occur all the entities involved in the transition must have a classification attribute. Thus the confidentiality controls

may be applied. These entities must also have a reference attribute in order to be addressable by an implementation. Nothing further is said about the particular addressing mechanism.

Note that not all entities need have trust, role or id attributes. These are only required by the active entities, ie those requesting transitions, and only then if they request security critical transitions. The particular separation of duty controls placed on entities is application specific. However, note that the set of conflicting roles for separation of duty should be non-empty, otherwise, unfaithful entities acting on their own could potentially alter the controls. Thus the following schema insists that all the entities involved in a transition have a classification, reference and at least one conflict attribute. Similarly a transition must preserve these properties. Nothing is said about entities destroyed by a transition except that they had the necessary attributes beforehand.

Correctness
TRANSITION
$before \subseteq dom\ s.Class \quad \wedge \quad after \subseteq dom\ s'.Class$ $before \subseteq dom\ s.Ref \quad \wedge \quad after \subseteq dom\ s'.Ref$ $before \subseteq dom\ s.Conflict \quad \wedge \quad after \subseteq dom\ s'.Conflict$ where $involved == r?.requestors$ $\cup r?.observed$ $\cup dom(s.Class \uparrow s'.Class)$ $\cup dom(s.Trust \uparrow s'.Trust)$ $\cup dom(s.Role \uparrow s'.Role)$ $\cup dom(s.Id \uparrow s'.Id)$ $\cup dom(s.Conflict \uparrow s'.Conflict)$ $\cup dom(s.Ref \uparrow s'.Ref)$ $\cup dom(s.Other \uparrow s'.Other)$ $before == involved \cap entities\ s$ $after == involved \cap entities\ s'$

Thus, the first aspect of integrity basically ensures that entities have the correct attributes in order that the confidentiality controls may be applied. The second aspect of integrity is that these control attributes are in some way appropriate to the information they protect. This is divided into two parts. The first concerns itself with modifications to controls of existing entities and the second with the controls given to new entities.

Separation of duty is used as the means to ensure the appropriateness of changes to security controls of existing entities. The *Separation_of_Duty* axiom below states that whenever any controls were modified by a transition there were sufficient requestors with the necessary conflicting roles, and further that these requestors were acting on behalf of an appropriate number of different human users, ie they possessed the *faithful* trust attribute, and had different ID attributes. It is important to note that because of possible collusions amongst the roles, separation of duty does not in itself guarantee security. However, sensible use of the mechanism does provide the best control that can be realistically achieved.

Separation of Duty
TRANSITION

$$\begin{aligned} & \forall e : \text{modified_controls} \bullet \\ & \quad \exists f : ID \rightarrow ROLE \mid f \subseteq s.Id^{-1} \parallel (\text{faithful_requestors} \Downarrow s.Role) \bullet \\ & \quad \downarrow f^{-1} \parallel (\# _) \in (s.Conflict \parallel \text{conflict_roles}) \parallel \{e\} \parallel \\ & \text{where} \\ & \quad \text{modified_controls} == \text{entities } s \cap \text{entities } s' \cap (\text{dom}(s.Class \uparrow s'.Class) \\ & \quad \cup \text{dom}(s.Trust \uparrow s'.Trust) \\ & \quad \cup \text{dom}(s.Role \uparrow s'.Role) \\ & \quad \cup \text{dom}(s.Id \uparrow s'.Id) \\ & \quad \cup \text{dom}(s.Conflict \uparrow s'.Conflict) \\ & \quad \cup \text{dom}(s.Ref \uparrow s'.Ref) \\ & \quad) \\ & \quad \text{faithful_requestors} == \{ r : r?.requestors \mid \text{faithful} \in s.Trust \parallel \{r\} \parallel \} \end{aligned}$$

The second aspect of appropriateness concerns itself with the controls given to new entities. The *Trusted Creation* axiom below states that whenever entities are created by a transition, at least one of the requestors was trusted to correctly set up the controls, ie possessed the *creator* trust attribute. Note that this trust covers the appropriateness of the controls that are given to the entity and also ensures that no necessary controls are omitted. Also note that the classification given to new entities is constrained by the *no flows down* axiom which insists that it be at least as high as the highest observed information. Establishing the basis of creator trust is application specific, and could, for example, also be enforced using separation of duty controls.

Trusted Creation
TRANSITION

$$\text{entities } s' \setminus \text{entities } s \neq \{ \} \Rightarrow \text{creator} \in s.Trust \parallel r?.requestors \parallel$$

Thus, ensurance of the appropriateness of control attributes is the combination of separation of duty controls upon any modifications and trust upon creation.

Appropriateness \triangleq *Separation of Duty* \wedge *Trusted Creation*

Integrity is seen to be the combination of the two aspects of correctness and appropriateness.

Integrity \triangleq *Correctness* \wedge *Appropriateness*

Finally, security is defined to be the confidentiality of information together with the supporting integrity of the controls that are used to enforce the confidentiality.

Security \triangleq *Confidentiality* \wedge *Integrity*

4. Examples of the Use of the Model

This section illustrates the use of the model by showing how to model some of the features commonly found in secure systems.

4.1. A User Interface: Windows

The human users of a computer system interact with it using the windows of a display. These windows are therefore modelled as the active entities of the system. In order to carry out the instructions of the human sitting at the display, the software which controls the window requires their identity, roles and clearance.

The user's initial window will be created by the login software, which is modelled as a highly trusted active entity acting on behalf of the systems owners. The login software will first check a login request against the data base of valid user identities, passwords, clearances and roles which form part of its functionality attributes. The actual classification given to the initial window of a user is application specific, but should never be higher than the human user's clearance. For example, some applications may allow the users to choose a session level lower than the clearance, or terminals situated in less well protected environments may not be allowed to run highly classified sessions.

In general, the initial window given to a user will form part of a Trusted Path, for example that of [Wisemanetal88]. This means that the window software is known to be functionally correct and to contain no additional functionality, ie it does exactly what the human user tells it to and contains no Trojan Horses. Such a window may be given the *faithful* and *dont_signal* trust attributes of the model. Should the initial window not be so trusted, there will need to be some way to establish a Trusted Path, for example by using an alert mechanism, to a *faithful* entity, so that security critical operations can be carried out.

From their initial window the users will wish create other windows, call operations and run application programs. Some of these operations and programs will continue the Trusted Path, in other words, they are also known to be functionally correct and free from Trojan Horses. However, a large part of the application specific software will not be trusted, as this can be prohibitively expensive and difficult.

Running programs is modelled as the creation of a new window entity. Note that in many cases there will be no visible representation of these window entities on the terminal screen. Whenever trusted software is called by trusted software the new window entity may be given the *faithful* and *dont_signal* attributes of the model. In other words, windows possessing these attributes, and the *creator* trust attribute, may create new window entities also with these attributes. Obviously, if untrusted software is called, the new window entity will not be given any trust attributes, and windows without these attributes may not cause new entities to be created which have them. The completion of an executing program is modelled by the destruction of the particular window entity, and control can be returned to the calling window entity. Thus, users may log on to the system and make a series of excursions off the Trusted Path as they run various untrusted application programs. In between these untrusted excursions they are able to perform trusted security critical operations.

In summary, the active entities of a system are modelled as windows with classification attributes representing the clearances of the human users. The confidentiality controls therefore prevent the window software from receiving and displaying information the human user is not cleared for, and the trust attributes of the window limit the activities of the controlling software. Consequently, the integrity of these controls is vital to the secure operation of the system. For this example, the integrity requirement is that the initial controls are appropriate to the human (guaranteed by the trusted login software) and that the controls are not alterable. Ensuring that the controls of any entity are unalterable is best specified as requiring the agreement of all possible roles, which includes the systems owners.

4.2. Contents of Objects

The contents of an object are obviously application specific, however, in general, objects may contain both character text and references to other objects. A naive attempt at modelling contents would probably be to have attributes representing each character and the various reference attributes, all as part of the functionality of the entity modelling the object. The problem with having separate attributes representing each character and reference is that the model is unable to notice changes in, say, the order

of these, as the *Other* relation still supplies the same set. Changes in order could potentially be used as a signalling channel. However, this is not a problem with the model but with the level of abstraction used. In the most abstract specification of functionality there ought to be a single attribute to represent the contents of an object. Various functions may be defined, as and when required, to supply aspects of the representation of these contents attributes. For example, a function can be defined which acts on a contents attributes and supplies the set of references it contains.

Briefly returning to the specification of window entities, they will have a single functionality attribute representing their contents, ie the character text and references to the objects that the window software may address. How the contents are displayed to the user is the concern of lower levels of abstraction. All that is required at the initial level is to be able to notice any changes and to ensure that the security axioms are upheld. Note that it is important to bear in mind that the purpose of an initial specification is to be as abstract as possible and to avoid making any design or implementation decisions.

Thus, there are attributes to represent every possible combination of characters and references (which is easily said in the infinite world of mathematical sets). Any alteration to the contents of an object is modelled by replacing its contents attribute with a different one.

4.3. Regradable Documents

This section considers an example requirement of regradable classified documents. As the text of a document may only be observed by sufficiently cleared users, their existence and classification is not considered to be classified. Thus, documents are modelled as passive entities, with a contents attribute, as above, as part of their functionality. For this example, security officer and user roles have been identified as providing the necessary separation of duty on the appropriateness of a regrade to a document. Therefore these are the roles that will be supplied by the conflict attributes of the formal model.

At the highest level of abstraction, regrading a document is specified as entities acting on behalf of a user and a security officer, simultaneously agreeing that the new classification is appropriate. The *Separation of Duty* axiom insists that these entities are carrying out the wishes of a human user, ie both possess the *faithful* trust attribute, and, as the controls of an entity are altered, the *no_signalling* confidentiality axiom insists that both requestors also possess *dont_signal*. In other words, document regrades must be performed from a Trusted Path.

In an implementation, this agreement to regrade is not simultaneous, but will first be requested by a user and then authorised, possibly at a much later date, by a security officer. Therefore, the implementation will introduce extra structure to the functionality of documents to record the request. However, so long as this extra structure is hidden, the security properties reasoned about at the higher levels still hold. For example, it must not be possible for another entity to be able to tell that there is a request to regrade pending, as this would introduce a new flow of information from the originator of the downgrade request to this other entity.

Thus, the model provides the means to ensure that regrades to a document are instigated by authorised human beings and not by software they may be executing, and further that these people are not colluding. However, even if these separation of duty controls have been set up appropriately, there is still a danger of downgrading information inappropriately.

For example, a document might be highly classified because of the content of only one paragraph or section. A suitably cleared user could delete this information, using an untrusted word processor, and request a downgrade to a lower classification in order that lower cleared users may read the remainder of the information. A small and simple Trojan Horse in this software could easily remember deleted information and just not display it. The document could be downgraded by the security officer, and the highly classified information is now available to unauthorised users. Thus, the separation of duty controls have been insufficient and an inappropriate regrade has been performed.

This problem can be solved by requiring that the document is reviewed using trusted software which displays all the information. Such a program would not be as difficult and expensive to trust as the word processor application as there will be no editing functions, only those to display the information. However, this does not completely solve the appropriateness problem, as even though it is guaranteed that all the information is displayed, the system cannot actually make the human security officer read it!

A more sophisticated Trojan Horse in a word processing application could encode any classified information it has access to directly into the text being edited. There are a number of ways it could do this; use the height of the spaces between the characters or words to encode the information; different fonts which look the same on the screen; fonts where the colour of the text is the same as the background; alter the spelling of words, eg encode 'ie' and 'ise' as a one and 'ei' and 'ize' as a zero, modulate double and single letters in the words; modulate punctuation; etc. It would take a vigilant security officer who was very good at spelling and grammar to notice the later few encodings.

Therefore, as well as a trusted display program, a canonical form would be required. However, should the system owners be this concerned about potential information leakages through downgrade, they should question whether they require downgrade at all, or whether the problem is that information is being initially over classified.

4.4. Journals and Audit

It is expected that most applications will require that information is journalled for audit purposes. The model does not supply any journalling mechanisms as the requirements are so application specific that anything built into the model would be either too general to be of any real use or too specific to be as generally applicable as the model hopes to be. This section considers how a few of the possible journalling requirements may be modelled.

The first example considered is where the system requires there to be a single journal to record all the interesting events. This is modelled as a passive entity, with functionality attributes representing the events. Since the journal is required to be updated whenever an attempt is made to access an object, the *no flows down* confidentiality axiom insists that the journal is classified at least as high as the highest observed entity, effectively system high or top. Consequently, it becomes very important to ensure that when the journal is updated, nothing whatsoever about its contents may be observed, ie the update is a 'pure write'. Note that the refinement to an implementation must preserve this property [Wiseman90]. The contents of such a journal may only be observed by requestors cleared to top.

Since the journal is used to record the operations of active entities of varying clearances and degrees of trust, it may be used as a signalling channel. For example, a Trojan Horse may encode a one or a zero through whether or not they choose to access a particular object. Therefore, even using separation of duty and the techniques discussed above, it can never be appropriate for the journal to be downgraded, and the information reviewed by anyone lower than top. However, an application could possibly permit journalled information to be reviewed by faithful entities, lower than top, as these would be acting on behalf of the human users who not not use signalling channels, or alternatively only by security officers. However, the information would first have to be copied and downgraded, so as not to violate the confidentiality axioms.

Should review by users cleared lower than top be required, the system could have a journal at each possible classification level. The events are then recorded in the journal at the classification of the active entities that caused them, thus upholding confidentiality. These journals may be reviewed by active entities classified at, or higher than, the entities whose actions are recorded. Therefore, there is no problem with the signalling channel, as the receivers are cleared for any information that might be encoded into the events of the journal.

The main reasons for journalling information are to make the human users accountable for their actions, and to alert the system owners to possible subverted users, penetrations of the system (often by guessing passwords) or Trojan Horse activity. Thus, the journalled information is user oriented. This, then, is a third approach to journalling, namely journals to record the actions of each user individually. To uphold the confidentiality axioms, these journals may be classified anywhere between system high, ie top, and the clearance of the user. As above, the particular classification chosen for these journals will depend upon the applications requirement for reviewing journalled information.

The final approach to journalling discussed here is object based. In the paper world, objects such as documents and files often maintain a list of the people who have accessed them on the front cover. A first attempt at modelling this journalled information would probably be as functionality attributes of the document or file entity. The mistake here is the inclusion of the journalled information as part of the attributes of the object entity, and not as a separate entity. For example, it is required that the open

operation supplies the textual contents of an object and records the event. Thus, in model terms, the object is both *observed* and *modified*, and consequently the *no_flows_down* confidentiality axiom would only allow users to open objects classified exactly at their clearance, and never lower. This is not the functionality that is required of a multi-level environment.

Therefore, the journals for objects also need to be modelled as entities in their own right. The functionality attributes of objects then include references for journal entities. It is important to note that the high level requirement of, for example, documents has been modelled by more than one entity, namely both document and journal entities.

Depending on the requirement for review, there may be a single journal classified top per object, or a journal for each classification able to access the object (essentially a multi-level journal). It is important point to note that in either case the journalled information concerning an object is effectively part of that object and is accessed via the entity modelling the object. Therefore, such a journal cannot be used to record unsuccessful attempts to access an object. This requirement would have to be met by one of the other approaches to journaling. Note that functionality specifications may freely mix the various types of journal.

It is interesting to note that this model has identified an apparent insecurity in the paper world, where, for example, all the access to a document are recorded on its cover and consequently may be reviewed by practically anyone. This is generally not considered to be a problem because of the incredibly low bandwidth of the paper world signalling channel, plus the fact that subverted human beings have much better ways of gaining classified information.

Should an application require alarms, operations could easily be specified which alerted a security officer after a certain number of unsuccessful attempts to access an object or perhaps after a number of failed login attempts at the same terminal. A flexible journaling system which allowed the activities that are to be journalled to be alterable could also be modelled. Such a requirement would probably best be dealt with by requiring there to be separation of duty on any changes to the type of information journalled, to ensure that they are appropriate.

4.5. Drafting Documents

It is expected that applications will invariably require new objects to be created. For example documents may be drafted by one user and then, when ready, turned into classified documents and made available to other users. Most systems would like to be able to use commercial off the shelf word processors, spelling checkers, and so on, for such tasks, as the cost and effort involved in developing and trusting such software would be prohibitive. It is therefore necessary that there be some mechanisms to limit the activities of such software.

As discussed earlier, *untrusted* software is modelled as a window entity with *no trust* attributes. Consequently, it will be unable to signal any information it possesses to any other entities through the channels of creation, deletion and alteration of controls. Any channels through shared resources, etc, are at a lower level of abstraction, and will be considered at the appropriate time in the refinement from the requirement to an implementation. Therefore, the main consideration when drafting documents is that the classification given to the final document is sufficient to protect the information it contains.

Session levels are one method of achieving this, for example where the user logs in at a certain classification level, and may only create objects at that level. However, this approach generally leads to information being over classified, with the consequent problems of downgrade, as discussed above. What is required is that the activities of the untrusted software editing the text are monitored, the worst case assumed, and the final classification of the document must not be lower than the highest classification of information that the software has had access to whilst drafting the document. This can be achieved using a floating label or high water mark [Woodward87].

Draft documents are modelled as entities classified at the clearance of the user creating them, and with an extra classification attribute as part of the functionality which acts as a high water mark. The *no signalling* confidentiality axiom will require the initial creation of the draft document to be performed from a Trusted Path, and the *Trusted Creation* and *no_flows_down* axioms require that the software is trusted to correctly classify it at the clearance of the user and to give it appropriate integrity controls. Operations which allow the untrusted software access to classified information, such as the

open document operation, must be trusted to always increase the high water mark classification. Subsequently, when the draft is being turned into a classified document, the requested classification can be checked against the high water mark classification to ensure that it is not too low.

Note that users generally create documents at or lower than their clearance, and therefore, giving a draft document its final classification is effectively a downgrade from the clearance of the user to this final classification. The separation of duty controls on this downgrade are the human user, who must be on a Trusted Path since the axioms require the requestor to possess both *faithful* and *dont_signal* trust attributes, and the trusted (and evaluated) high water mark software, which is effectively acting on behalf of the system owners.

4.6. A Multi-level Directory

Some systems may require a directory in which files of various classifications may be stored. A directory listing will supply the names and classifications of all files at or below the requesting user's clearance. Users can never discover the existence of files classified higher than their clearance. Thus in this case, the classification protects both the name and the text of a file. Should a user request the creation of a file with the same name as an existing higher classified file, the request will not be rejected as the user would then know classified information, ie the name of a file they are not cleared for. Instead, the new file will be created with the duplicate name plus some indication of its classification in order that the files cannot be confused.

In this case knowledge of the existence of files is classified, and therefore the files themselves cannot be modelled as entities. Instead a filestore entity is created for each classification which a file in the directory may be given. The functionality attributes of these filestore entities will represent the names and texts of all the files at that particular level. Thus a multi-level directory may be modelled by a collection of these filestore entities, one at each of the required classification levels.

The operations are modelled as requested by active entities, such as the window entities described above. The *no_flows_down* confidentiality axiom will prevent these entities from observing anything about filestore entities classified higher than them. One consequence of this is that a requestor will be unable to distinguish between "file does not exist" and "not cleared to see file" error messages. These active entities do not have to be trusted, as although they could use the existence of files and their names to signal any information they know, the information may only be received by entities able to access the original information anyway, ie at or above the classification of the signalling entity.

Since, filenames are only unique within a classification level, the specification has to decide which of the possible options for each of the operations is required. For example, it has to be decided when asking to open a file whether i) the most classified file with the given name that the requestor is cleared to is opened, ii) all the files with that name are presented and the requestor indicates which is required, or iii) the requestor initially presents both a classification and a file name.

When creating a new file, the requestor would normally expect to receive confirmation that the operation was successful. Therefore, as well as ensuring that the requestor is unable to modify a filestore entity (ie create a new file in it) classified lower than itself, the *no_flows_down* confidentiality axiom would also prevent it modifying filestores classified higher. Hence in this example, requestors may only write at their clearance.

The directory listing operation could cause the requestor to be modified with the names of all the files in filestore entities at or below their clearance. As the names are not necessarily unique, the classification of the files could be supplied as well. Alternatively, the operation may wish to only list those files at the clearance of the requestor.

Since none of the security controls are altered there is never a requirement for the active entities to possess *faithful* or *dont_signal* trust attributes. However, should an application require, for example, that files may only be modified by entities faithfully acting on behalf of a human user, this constraint may be easily added to the functionality specification.

Note that this example has modelled the logical separation of information. This may be refined either to an implementation using separate address spaces, or to one where the addressing mechanism is highly trusted.

4.7. Access Control Lists

Access control lists are often provided by secure systems as a means of providing need-to-know controls over and above the confidentiality controls. This section considers two approaches to the provision of such a mechanism. However, it is important to remember that the exact nature of an access control mechanism is application specific and the examples given here are not expected to satisfy all requirements.

Firstly, access control lists may be modelled as an additional control on the outside of the boxes. Since the model does not permit the security controls to be classified, this would mean that the access control lists were freely observable. Another consequence would be that alteration to the lists would be subject to separation of duty controls to ensure that they were not changed inappropriately. For example, granting a particular right would generally require more than one requestor, all possessing the *dont_signal* and *faithful* trust attributes, and care would have to be taken that the confidentiality controls were not changed inappropriately at the same time. However note that, should an application require, a slightly different model could be developed with a different set of controls on the alteration of the access control mechanism than on the confidentiality controls.

Secondly, the access control lists can be viewed as additional functionality attributes of the entities they protect, ie inside the black boxes of the analogy. Confidentiality therefore prevents access control lists from being observed by any requestors not cleared to observe the information they protect, but only ensures that alterations to the lists do not violate the no flows down aspect of confidentiality.

As an example of this consider two access rights, namely the right to **observe** and the right to **modify** the contents of an object, and four operations. The operations are those to observe and modify the contents of an object, and those to add both types of right to the access control lists of an object. However, as the confidentiality controls still apply, even entities with the observe right will never be able to observe the contents of another entity if they are not also suitably cleared.

These access control lists may be modelled by introducing a set of attributes to represent all possible sets of user identities for each access right. Functions may be defined which supply the particular set of identifiers associated with one of these attributes. Entities requiring an access control mechanism will be given an attribute representing the set of users with observe access and another representing the set of users with modify access. Remember that the observe and modify access control applies to all the functionality attributes except those implementing the access control mechanism, as no control may be used to protect itself. The functionality attributes of such entities may therefore be viewed as split into two, namely an access control mechanism and the information it protects.

Upon an initial consideration it would appear that pure modifications, ie no observation of the contents, to objects with an access control lists would be possible, as it is allowed by the model (and useful when, for example, journalling information). This would mean that entities cleared at or lower than an object may modify its contents. The confidentiality controls obviously exclude highly cleared requestors from modifying lowly classified entities. However, by succeeding to modify the object, the requestor has effectively observed part of its contents, namely, the fact that it is in the appropriate access control list. Therefore, in order to uphold confidentiality requestors may only modify an object at their clearance.

Lastly, we consider two example requirements for the granting an access right to another user, ie modifying the access control list. The first is where a user may grant a particular right for an object only if they have that right themselves. Secondly, granting an access right requires two users to agree, both of whom possess that right, ie there is separation of duty.

As for modifying the contents of an object, requestors attempting to grant a right to another have to be cleared exactly at the classification of the object, as they are both modifying the object, and observing whether they themselves have that right. There are no constraints on the clearance of the user who is being granted the right as the confidentiality controls do not allow low users to discover that they been granted the right. Note that this model does not require the entities modifying the access control lists to

be trusted, as the trust in the model is only concerned with the confidentiality controls. However, it can easily be specified that for certain operations, such as granting or removing a right, the requestors must be acting on behalf of a human user, ie possess the *faithful* trust attribute.

Where there is separation of duty on the granting of an access right, this is modelled as a simultaneous agreement between the two requesting entities. As in the regradable document example, this simultaneous agreement will be implemented as a request followed by an authorisation, and extra structure will be added to a entity to accommodate this. This structure must not be visible to any other entities, otherwise it could be used as a signalling channel.

5. The Terry-Wiseman Approach to High Assurance Implementations

This section provides an overview of the role of the Terry-Wiseman Security Model in the procurement of high assurance, high functionality computer systems, where the confidentiality of information is a vital concern. The general approach to the achievement of high assurance implementations is also discussed.

As illustrated in the previous section, the Terry-Wiseman Model provides a framework to define the security requirements of a system in terms of the objects and operations which are controlled by the security policy. The use of the model in discussions between the customers, developers and evaluators of a secure system can ensure that they all interpret the requirements in the same way, and consequently can agree that the statement of the security they require is valid and has been met by the system produced. This is essential for there to be any reasonable level of assurance that a system intended to be secure, is indeed so.

Using Terry-Wiseman to model the functionality requirements for a secure system clearly identifies any aspects of the desired functionality that are contradictory to the statement of security and any potential covert channels. This enables all concerned to analyse the vulnerabilities and assess the risks, and agree any changes to the functionality that may be necessary, evaluating the consequences of possibly several alternatives, all in the initial stages of a project. This is obviously more cost effective than discovering problems as an implementation proceeds or analysing the software at the end for covert channels and attempting to fix them in a more ad hoc manner.

Thus, the first stage in the procurement of a secure system should be to model the functionality requirements in terms of entities and attributes and to describe the operations on each object in terms of sequences of secure state transitions. Should it not be possible to find a way of modelling the objects and operations in terms of the model, it may be that at least part of the requirement is fundamentally insecure. Should this be the case, the places where the problems were encountered will have been explicitly identified, and the developers can present the customers with suggestions as to how the requirement may be made secure, hopefully without effecting the functionality a great deal. Should the customers not wish to change the functionality at the point of the security problem, or not perceive the problem to be a particular threat, then at least it is known exactly what security is being sacrificed.

This modelling stage will result in a formal specification of the objects and operations of the required system in terms of entities, attributes and sequences of state transitions. The next stage is to refine the specification towards an implementation, for example refine sets into sequences and the multiple requestors into sequences of single requestor transitions. All refinements introduce extra detail, and care must be taken that this detail is not used for covert information flows, by-passing the security properties reasoned about at the higher level. Preserving the abstract interface, ie hiding the extra detail, is one method of avoiding these problems. Briefly returning to the analogy, this requires that any detail introduced in the implementation of the contents of black boxes, is not visible outside the boxes. However, should it not be possible, or desirable, to hide the extra detail, the new information flows will need to be analysed.

The Terry-Wiseman model essentially requires two properties of the implementation; 1) immutable attributes, ie tamper-proof objects, and 2) a way to ensure that whenever an entity is modified or observed the state transition axioms are upheld. Four basic mechanisms have been identified as sufficient to implement these requirements; unforgable opaque addresses, data hiding, pedigree and context. [Harrold89] provides an overview of these mechanisms.

The Terry-Wiseman Approach is to use these four mechanisms as an intermediate step in the refinement of the requirements of an application in terms of the model to its actual implementation. Consequently, the designers of a secure application do not need to concern themselves with the minute detail of the facilities of the target machine while working at the higher levels of abstraction. The realisation of the mechanisms on any particular hardware is of course concerned with fine details, but need only ever be considered once for each target. More importantly, splitting the implementation process into steps in this way, not only simplifies it, but will also make secure applications more portable.

[Wiseman90] shows how the elements of the Terry-Wiseman model can be mapped onto the four mechanisms. [Wiseman90] also considers the proof obligations that arise from the refinement of model to mechanisms and explores some implementation techniques.

6. Evolution of the Model

This section identifies the differences between the original formalisation of the Terry-Wiseman Model [Terry&Wiseman89] and the model presented in this paper. The reasons the changes were considered necessary are also given. However, in summary, the majority of the changes resulted from an exercise in modelling a demonstration system [Harrold90], and are mainly of an interpretational nature.

The most notable addition to the model has been to provide references for entities. This came about when modelling functionality was considered in detail, since whereas security is only concerned with seeing that the axioms are upheld, functionality specifications are concerned that the secure transitions act upon the correct entities. Initially we tried adding references as part of the functionality attributes of the state. However, we soon came to realise that the properties of references effectively make them a control. This is mainly because if references are solely functionality, the check that the correct entity is being addressed cannot be performed without opening up the entity to compare references. Thus, references have been made a control.

The problems with modelling references caused us to consider what it actually meant for something to be a control. An important conclusion we came to was that observing the controls on an entity should not invoke the protection mechanism, as these controls are part of the protection mechanism and consequently cannot be used to protect themselves. Thus, the model presented here stresses that in a transition the observed entities are all those, and only those, whose functionality attributes contributed in any way to modifications to the state. The controls of an entity are all assumed to be visible.

As a result of this stronger interpretation of the properties of controls, the *no_signalling* axiom (*confidentiality2* in the original formalisation) now ensures that none of the controls can have classified information written or encoded into them. Previously, it only ensured that the classification and conflict attributes were not classified. Similarly, the *no_flows_down* axiom (essentially *confidentiality1* from the original formalisation) is now concerned only with observation of the *Other* attributes. However, *no_flows_down* has also been strengthened, and does not allow information to flow down into newly created entities, nor into entities which are regraded by the transition. In practice, the *new_class* trust attribute from the original formalisation would have ensured that the information in newly created entities was correctly classified. Also, the separation of duty controls would have prevented inappropriate downgrades. However, we decided that the much stronger definition of this central principle of the model, namely that information cannot flow downwards, was more appropriate. The model now ensures that the only way information may flow down is through an explicit regrade.

The new formalisation of the model has much more realistic separation of duty controls. The previous controls were too simplistic as they did not allow a number people with the same role to agree to modifications to controls, people to have more than one role, or more than one set of conflicting roles to be specified. Consequently, *CONFLICT* attributes have been identified and the *Role* and *Conflict* aspects of the state have been amended accordingly. Also, *ID* attributes and the *Id* function have been included so that the *Separation of Duty* axiom can ensure that there is an appropriate number of different people requesting a modification to controls.

The original formalisation of the model insisted that all requestors of a transition were at the same classification level, although this fact could easily be overlooked. However, what we actually required was that the requestors may be at any level, but that the higher ones did not base their decision as to whether to agree to a transition on any highly classified information. In other words, we required the ability to have unobserved requestors of a transition. Similarly, we did not want to insist that the requestors of a transition were always modified. In other words, requestors could agree to their part of a transition but not necessarily know whether the complete transition took place. Consequently the decision, *d'*, which effectively meant that all the requestors knew whether or not the transition took place, was removed from the transition. Instead, any result must explicitly be made part of the state. In the new interpretation, an unmodified requestor does not know whether the transition took place.

One of the most noticeable differences is in the presentation of the model. In all previous formalisations of the model the valid state transition definition included global definitions of the sets of entities that 'things happened to'. As mentioned above, we found when using the model in earnest that several of these definitions were not quite as had been intended. However, it was difficult to understand the implications of the definitions as their use was so far away from their introduction. Thus, the model

presented here makes greater use of local definitions, namely the 'where phrase', to identify these important sets only at their point of use.

The Correctness axiom in this paper is essentially the *Integrity* axiom from the previous formalisation. However, it has been extended to cover reference and conflict attributes as well as classification attributes.

Another change has been the replacement of the separate trust attributes for setting up each aspect of the security controls on new entities by the single *creator* trust attribute and the *Trusted_Creation* axiom. Separate attributes were originally included in an attempt to ensure that the creator always provided appropriate values for each control. This was erroneous thinking as in fact there is no advantage in having separate trust attributes for each control.

Other minor changes are the definition and use of the *flatten* operation which supplies all the relationships that comprise a state. Also, some definitions have been renamed to make their meaning more obvious, and the schemas which had Bell and LaPadula type names have been given more descriptive names.

7. Summary and References

This paper has presented the Terry-Wiseman Security Policy Model. This is a formal model of security which enforces the confidentiality of information, and also provides the means to ensure that the confidentiality controls have integrity.

We believe that this model is applicable as a statement of the security requirement in a wide range of high functionality computer applications. However, the validity of any statement of security is entirely a matter of human judgement and requires that all concerned understand and interpret the model in the same way. Therefore, this paper has also described the context and assumptions of the Terry-Wiseman Model using a simple analogy, and provided a variety of examples of its use.

In addition, an overview of the role the Terry-Wiseman Model can play in the procurement of high assurance, high functionality computer systems has been presented.

References

- [Bell74] Secure Computer Systems
D E Bell
MTR-2547, Volumes 1, 2 and 3, Mitre Corp., April 1974
- [Bottomley88] SMITE - RSRE's Computer Architecture for Multi-level Security
P C Bottomley, S R Wiseman
Proc of Milcomp 1988, pages 25 - 30.
- [Craig87] m-Eves: A Tool for Verifying Software
D Craigen, S Kromodimoeljo, I Meisels, A Neilson, B Pase, M Saaltink
IPSA Conference Paper CP-87-5402-26, October 1987
- [Goguen82] Security Policies and Security Models
J A Goguen, J Meseguer
Proc of the IEEE Symposium on Security and Privacy, Oakland CA, April 1982
pages 11 - 20
- [Harrold89] An Introduction to the SMITE Approach to Secure Computing
C L Harrold
Computers and Security Journal, vol 8, October 1989, pages 495 - 505
- [Harrold90] An Example System Specified Using the Terry-Wiseman Approach
C L Harrold
RSRE Report, in preparation.
- [Linden76] Operating System Structures to Support Security and Reliable Software
T A Linden
NBS Technical Note 919, August 1976
- [Spivey88] The Z Notation: A Reference Manual
J M Spivey
Prentice-Hall International 1988. ISBN 0-13-983768-X
- [Terry90] Proof of the SMITE Model
P F Terry
CRL-2711-03, Capella Research Ltd, January 1990
- [Terry&Wiseman89] A 'New' Security Policy Model
P F Terry, S R Wiseman
Proc of the IEEE Symposium on Security and Privacy, Oakland CA, May 1989
pages 215 - 228

- [Wiseman86] A Secure Capability Computer System
S R Wiseman
Procs of the IEEE Symposium on Security and Privacy, Oakland CA, April 1986
pages 86 - 94
- [Wisemanetal88] The Trusted Path Between SMITE and the User
S R Wiseman, P F Terry, A W Wood, C L Harrold
Procs of the IEEE Symposium on Security and Privacy, Oakland CA, April 1988
pages 147 - 155
- [Wiseman90] Basic Mechanisms for Computer Security
S R Wiseman
RSRE Report 89024
- [Woodward87] Exploiting the Dual Nature of Sensitivity Labels
J P L Woodward
procs of the IEEE Symposium on Security and Privacy, Oakland CA, April 1987
pages 23 - 30

The author would like to acknowledge the considerable contributions of Simon Wiseman and Phil Terry to the technical content of this paper. Thanks are also due to Peter Bottomley, Chris Cant and Gill Randell for their comments on earlier drafts.

Annex: An Overview of the Z Notation

Z is a powerful mathematical notation that has been developed by the Programming Research Group at Oxford University. The underlying basis of Z is standard set theory, and it makes use of the associated notation. Properties about sets are described using predicate calculus. A Z specification is structured into self contained parts using schemas.

New sets are introduced between square brackets, for example, [BOOK] introduces the set of all possible books, or using == followed by a definition of the set.

{}	empty set
\in	LHS is a member of the set on the RHS
\notin	LHS is not a member of the set on the RHS
\subseteq	set on the LHS is a subset of the one on the right (possibly equal)
\subset	set on the LHS is a proper subset of the one on the right (never equal)
\cup	result is the union of the two sets
\cap	result is the intersection of the two sets
\setminus	result is the set equal to the LHS with members of the RHS removed
\neq	inequality
$\#$	number of elements in a set
\mathbb{N}_1	the set of strictly positive natural numbers
$x : T$	declaration, x is an element drawn from the set T
$\{ D \mid P \}$	the set of values from the declarations D such that the predicate P holds
\mathcal{P}	powerset, ie the set of all possible subsets of a particular set

Thus, *childrens* : $\mathcal{P} \text{BOOK}$, says that the set of books for children is 'drawn from the set of subsets' of the set BOOK, ie childrens books are a subset of all books.

A relation may be viewed as a set of ordered pairs. Functions are a special type of relation where there is a single element in the range for each element of the domain. Thus the operators defined for sets are applicable to both functions and relations.

<i>dom</i>	domain of a relation, ie all the first elements of the ordered pairs
<i>rng</i>	range of a relation, ie all the second elements of the ordered pairs
\leftrightarrow	relation
\rightarrow	total function, domain is all possible members of the set
\Rightarrow	partial function
\Rightarrow	injective partial function, each element in range associated with only one in domain
\mapsto	maplet, graphic way of expressing an ordered pair
$[]$	relational image, $R[S]$ the set related by the relation R to the members of the set S
R^{-1}	inverse of the relation R
$S \upharpoonright R$	domain restriction, restrict relation R to those domain elements in set S
$S \circ R$	relational composition, ie S followed by R (type of rng S must equal type of dom R)

<i>declaration</i>	an axiomatic definition, the declarations are global
<i>predicates</i>	the predicates define properties about them

<i>name</i>	a schema, the signature declares some variables and their types
<i>signature</i>	the predicates define properties about them
<i>predicates</i>	the objects declared in one schema are made available to another by including the name of the schema in the signature

$s.t$	project the variable t from the schema variable s
\triangleq	schema definition
\wedge	and
\Rightarrow	implication
$\forall x : T \bullet P$	for all x of type T predicate P holds
$\exists x : T \mid D \bullet P$	there exists an x of type T for which predicates D and P hold
<i>Predicates</i>	Where clause, shorthand for $\exists D \bullet P$
<i>where</i>	
<i>Declarations</i>	

The following conventions are used for variable names in schemas representing operations:

<i>undashed</i>	state before
<i>dashed, ie ending in '</i>	state after
<i>ending in ?</i>	inputs or parameters
<i>ending in !</i>	outputs or results

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known)

Overall security classification of sheetUnclassified.....
 (As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).)

Originators Reference/Report No. REPORT 90001		Month MARCH	Year 1990
Originators Name and Location RSRE, St Andrews Road Malvern, Worcs WR14 3PS			
Monitoring Agency Name and Location			
Title THE TERRY-WISEMAN SECURITY POLICY MODEL AND EXAMPLES OF ITS USE			
Report Security Classification Unclassified		Title Classification (U, R, C or S) U	
Foreign Language Title (in the case of translations)			
Conference Details			
Agency Reference		Contract Number and Period	
Project Number		Other References	
Authors HARROLD, C L			Pagination and Ref 24
Abstract This paper presents a model of security for computer systems where it is essential that the confidentiality of the information is maintained. The model is introduced using a simple analogy, it is then defined formally and its use illustrated by examples. The framework for using the model to achieve high assurance implementations of high functionality systems is also discussed.			
			Abstract Classification (U, R, C or S) U
Descriptors			
Distribution Statement (Enter any limitations on the distribution of the document) Unlimited			